An n log n Algorithm for Deterministic Kripke Structure Minimization

Karl Meinke, Muddassar A. Sindhu

School of Computer Science and Communication, Royal Institute of Technology 10044, Stockholm, Sweden.

Abstract

We introduce an algorithm for the minimization of deterministic Kripke structures with $O(kn \log_2 n)$ time complexity. We prove the correctness and complexity properties of this algorithm.

1. Introduction

The problem of minimizing automata and transition systems has been widely studied in the literature. Minimization involves finding the smallest equivalent structure, using an appropriate definition of equivalence, (e.g. language equivalence or simulation equivalence). In many software engineering applications, automata need to be minimized before complex operations such as model checking or test case generation can be carried out.

For different automata models and different notions of equivalence, the complexity of the minimization problem can vary considerably. The survey [1] considers minimization algorithms for DFA up to language equivalence, with time complexities varying between $O(n^2)$ and $O(n\ log\ n)$. Kripke structures represent a generalisation of DFA to allow non-determinism and multiple outputs. They have been widely used to model concurrent and embedded systems. An algorithm for mimimizing Kripke structures has been given in [2]. In the presence of non-determinism, the complexity of minimization is quite high. Minimization up to language equivalence requires exponential time, while minimization up to a weaker simulation equivalence can be carried out in polynomial time (see [2]).

By contrast, we will show that *deterministic* Kripke structures can be efficiently minimized even up to language equivalence with a worst case time complexity of $O(kn \log_2 n)$. For this, we generalise the concepts of right language and

Email address: karlm@nada.kth.se, sindhu@csc.kth.se (Karl Meinke, Muddassar A. Sindhu)

Nerode congruence from DFA to deterministic Kripke structures. We then show how the DFA minimization algorithm of [3] can be generalised to compute the Nerode congruence $\equiv$ of a deterministic Kripke structure $\mathcal{K}$. The quotient Kripke structure $\mathcal{K}/\equiv$ is minimal and language equivalent to $\mathcal{K}$. Our research [4] into software testing has shown that this minimization algorithm makes the problems of model checking and test case generation more tractable for large models.

The paper is organized as follows. In Section 2, we introduce some mathematical pre-requisites. In Section 3, we give a minimization algorithm for deterministic Kripke structures. In Section 4, we give a correctness proof for this algorithm. In Section 5 we provide a complexity analysis. Finally, in Section 6 we discuss some conclusions.

2. Preliminaries

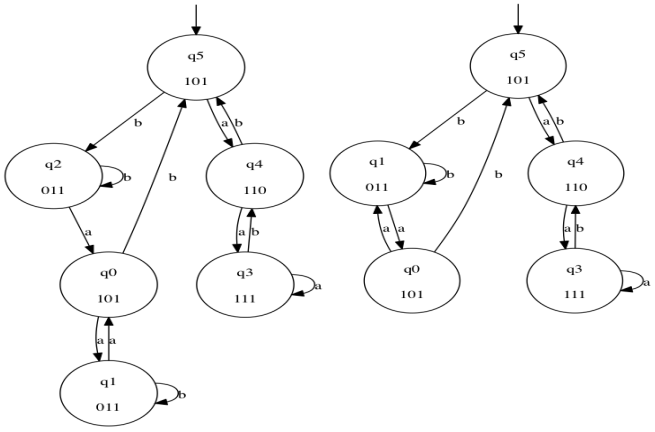
We assume familiarity with the basic concepts of deterministic finite automata (DFA). A *Kripke structure* is a generalisation of a DFA to allow multiple outputs and non-determinism. A Kripke structure $\mathcal{K}$ over a finite set AP of atomic propositions is a five tuple $\mathcal{K} = \langle Q, \Sigma, \delta, q_0, \lambda \rangle$, where Q, is the set of states, $\Sigma = \{\sigma_1, ..., \sigma_n\}$ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation for states, q_0 is the initial state of $\mathcal{K}$ and $\lambda : Q \to 2^{AP}$ is a function to label states. If $|AP| = k$ we say that $\mathcal{K}$ is a k-bit Kripke structure.

We say that $\mathcal{K}$ is *deterministic* if the relation δ is actually a function, $\delta : Q \times \Sigma \to Q$. We let $\delta^* : Q \times \Sigma^* \to Q$ denote the iterated state transition function where $\delta(q, \epsilon) = q$ and $\delta^*(q, \sigma_1, ..., \sigma_n) = \delta(\delta^*(q, \sigma_1, ..., \sigma_{n-1}), \sigma_n)$. Each property in AP describes some local property of system states $q \in Q$. It is convenient to redefine the labelling function λ as $\lambda : Q \to \mathbb{B}^k$ given an enumeration of the set AP. Then the iterated output function $\lambda^* : Q \times \Sigma^* \to \mathbb{B}^k$ is given by $\lambda^*(q, \sigma_1, ..., \sigma_n) = \lambda(\delta^*(q, \sigma_1, ..., \sigma_n))$. More generally for any $q \in Q$ define $\lambda^*_q(\sigma_1, ..., \sigma_n) = \lambda^*(q, \sigma_1, ..., \sigma_n)$. Given any $R \subseteq Q$ we write $\lambda(R) = \cup_{r \in R} \lambda(r)$. We let $q.\sigma$ denote $\delta(q, \sigma)$ and $R.\sigma$ denotes $\{r.\sigma \mid r \in R\}$ for $R \subseteq Q$.

We can represent a Kripke structure graphically in the usual way using a *state transition diagram*. For example, a Kripke structure with three bit labels in the output is shown in Fig 1(A).

2.1. Minimal DFA and minimal deterministic Kripke structures

Let us consider a DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$. For each state $q \in Q$ of $\mathcal{A}$ there corresponds a subautomaton of $\mathcal{A}$ rooted at q which accepts the regular language $\mathcal{L}_q(\mathcal{A}) \subseteq \Sigma^*$, consisting of just those words accepted by the subautomaton with q as initial state. Thus $\mathcal{L}_{q_0}(\mathcal{A})$ is the language accepted by $\mathcal{A}$. The language $\mathcal{L}_q(\mathcal{A})$ is called either the *future* of state q or the *right language* of q. $\mathcal{A}$ is *minimal* if for each pair of distinct states $p, q \in Q$, we have, $\mathcal{L}_p(\mathcal{A}) \neq \mathcal{L}_q(\mathcal{A})$. For any



(A) Non minimal Kripke structure K (B) Minimized Kripke structure K_{min}

states	a	b
q_0	q_1, q_2	-
q_1	q_0	q_1
q_2	-	q_2, q_5
q_3	q_3, q_4	-
q_4	q_5	q_3
q_5	-	q_0, q_4

(C) δ^{-1} function of Kripke structure K

Figure 1: 3-bit Kripke Structure $\mathcal{K}$

regular language $\mathcal{L} \subseteq \Sigma^*$ there is a smallest DFA (in terms of the number of states) accepting $\mathcal{L}$. This DFA is minimal, and is unique up to isomorphism.

An equivalence relation $\equiv$ can be defined on the states of a DFA by $p \equiv q$ if and only if $\mathcal{L}_p(\mathcal{A}) = \mathcal{L}_q(\mathcal{A})$. This relation is a congruence, i.e. if $p \equiv q$ then $p.\sigma \equiv q.\sigma$ for all $\sigma \in \Sigma^*$. It is known as the *Nerode congruence*. Consider the quotient DFA $\mathcal{A}/\equiv$. This is the unique smallest DFA which accepts the regular language $\mathcal{L}_{q_0}(\mathcal{A})$. The problem of minimizing a DFA $\mathcal{A}$ is therefore to compute its Nerode congruence, which will be the identity relation if, and only if $\mathcal{A}$ is a minimal automaton.

The problem of computing a minimal Kripke structure $\mathcal{K}$ is an analogous but more general problem. In this case, the right language $\mathcal{L}_q(\mathcal{K})$ associated with a state q of $\mathcal{K}$ can be defined by

$$\mathcal{L}_q(\mathcal{K}) = \{ (\sigma_1, ..., \sigma_n, a) \in \Sigma^* \times \mathbb{B}^k \mid \lambda^*_q(\sigma_1, ..., \sigma_n) = a \}.$$

As before, $\mathcal{K}$ is *minimal* if for each pair of distinct states $p, q \in Q$ we have, $\mathcal{L}_p(\mathcal{K}) \neq \mathcal{L}_q(\mathcal{K})$. There is again a smallest Kripke structure associated with a right language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{B}^k$. This Kripke structure is also minimal, and unique up to isomorphism. The Nerode congruence for a Kripke structure $\mathcal{K}$ is now defined by:

$$p \equiv q \text{ if and only if } \lambda^*_p(\sigma_1, ..., \sigma_n) = \lambda^*_q(\sigma_1, ..., \sigma_n) \text{ for all } (\sigma_1, ..., \sigma_n) \in \Sigma^*.$$

and $\mathcal{K}/\equiv$ is the unique smallest Kripke structure associated with the right language $\mathcal{L}_{q_0}(\mathcal{K})$. So the problem of minimising $\mathcal{K}$ is to compute this congruence.

3. Kripke Structure Minimization Algorithm

Algorithm 1 presents an efficient algorithm to compute the Nerode congruence $\equiv$ of a deterministic Kripke structure $\mathcal{K}$, which is the same as the state set of the associated quotient Kripke structure $\mathcal{K}/\equiv$. We demonstrate the behavior of this algorithm on a simple example given in Fig.1(A) as follows.

The algorithm begins by inverting the state transition table as shown in Fig.1(C). Then it creates four initial blocks of states on the basis of unique bit labels which are: $B_1 = \{q_0, q_5\}$, $B_2 = \{q_1, q_2\}$, $B_3 = \{q_3\}$ and $B_4 = \{q_4\}$. Next it is checked whether the number of blocks is equal to the number of states $|Q|$ of the given Kripke structure. This is not the case, so the next step is to refine each partition block B_i into subsets $B(\sigma, i)$ of states which have predecessors via each input symbol of $\sigma \in \Sigma$. This gives $B(a, 1) = \{q_0\}$, $B(b, 1) = \{q_5\}$, $B(a, 2) = \{q_1\}$, $B(b, 2) = \{q_1, q_2\}$, $B(a, 3) = \{q_3\}$, $B(b, 3) = \{q3\}$, $B(a, 4) = \{q_4\}$ and $B(b, 4) = \{q_4\}$. The next step is to initialize the waiting list $W(\sigma)$ for each symbol $\sigma \in \Sigma$ by inserting the block numbers of all non-empty subpartition blocks $B(\sigma, i)$ created in the previous step. We obtain $W(a) = \{1, 2, 3, 4\}$ and $W(b) = \{1, 2, 4\}$.

Now the algorithm can refine the initial partition $B_1, \ldots, B_4$ by iterating the loop on line 10 until $W(\sigma) = \emptyset$ for all $\sigma \in \Sigma$. For $i = 1$ and $a \in \Sigma$ we have $W(a) = \{2, 3, 4\}$ and $B(a, 1) = \{q_0\}$. We can see that $\delta(q_1, a) = q_0 \in B(a, 1)$ and $\delta(q_2, a) = q_0 \in B(a, 1)$. But both q_1 and q_2 are in B_2. Therefore $B'_2 \not\subset B_2$ and hence no refinement of the partition is possible in this step.

We proceed with the next iteration of the loop by deleting $i = 2$ from $W(a)$ so that $W(a) = \{3, 4\}$. Now we have $B(a, 2) = \{q_1\}$. We can see that $\delta(q_0, a) = q_1 \in B(a, 2)$. Therefore we have $B'_1 = \{q_0\}$. Since $B'_1 \subset B_1$ we therefore split B_1 into $B_5 = B_1 - B'_1 = \{q_0, q_5\} - \{q_0\} = \{q_5\}$ and $B_1 = B'_1 = \{q_0\}$. Next we update the subsets $B(\sigma, i)$ and we get $B(a, 1) = \{q_0\}$, $B(b, 1) = \{\}$, $B(a, 5) = \{\}$ and $B(b, 5) = \{q_5\}$. The updated waiting sets are then $W(a) = \{1, 3, 4\}$ and $W(b) = \{1, 2, 4, 5\}$. Next we choose $i = 1$, $\sigma = a$ and $W(a) = \{3, 4\}$ and we obtain $B(a, 1) = \{q_0\}$. It can be seen that $\delta(q_1, a) = q_0 \in a(a, 1)$ and $\delta(q_2, a) = q_0 \in a(a, 1)$. Therefore $B'_2 = \{q_1, q_2\}$, but $B'_2 \not\subset B_2$ and hence no refinement of the partition is possible in this case. We delete $i = 3$ from $W(a)$ and obtain $W(a) = \{4\}$ and $B(a, 3) = \{q_3\}$. We then find that for $q_4 \in B_4$, $\delta(q_4, a) = q_3 \in B(a, 3)$. Therefore we have $B'_4 = \{q_4\}$. But $B'_4 \not\subset B_4$, so no refinement of the partition is possible in this case. Continuing in the same way it will be seen that there is no further refinement of the partition possible for $i = 4$ and $\sigma = a$ and for $i = 1, 2, 4, 5$ and $\sigma = b$ both $W(a)$ and $W(b)$ become empty. We terminate with five blocks in the partition. These constitute the states of our minimized Kripke structure as shown in Fig 1(B).

Input: A deterministic Kripke structure $\mathcal{K}$ with no unreachable states and k output bits.

Output: The Nerode congruence $\equiv$ for $\mathcal{K}$, i.e. equivalence classes of states for the minimized structure $\mathcal{K}_{min}$ behaviourally equivalent to $\mathcal{K}$.

1 Create an initial state partition $P = \{B_q = \{q' \in Q \mid \lambda(q) = \lambda(q')\} \mid q \in Q\}$. Let $n = |P|$. Let $B_1, ..., B_n$ be an enumeration of P.
2 **if** $n = |Q|$ **then go to** line 30.
3 **foreach** $\sigma \in \Sigma$ **do**
4 **for** $i \leftarrow 1$ **to** n **do**
5 $B(\sigma, i) = \{q \in B_i \mid \exists r \in Q \ s.t \ \delta(r, \sigma) = q\}$. /*This constitutes the subset of states in block B_i which have predecessors through input σ. */
6 $count = n + 1$;
7 **foreach** $\sigma \in \Sigma$ **do**
8 choose all the subsets $B(\sigma, i)$ (excluding any empty subsets) and put their block numbers i on a waiting list (i.e. an unordered set) $W(\sigma)$ to be processed.
9 Boolean splittable = true;
10 **while** *splittable* **do**
11 **foreach** $\sigma \in \Sigma$ **do**
12 **foreach** $i \in W(\sigma)$ **do**
13 Delete i from $W(\sigma)$
14 **for** $j \leftarrow 1$ **to** $count - 1$ *s.t.* $\exists t \in B_j$ *with* $\delta(t, \sigma) \in B(\sigma, i)$ **do**
15 Create $B'_j = \{t \in B_j \mid \delta(t, \sigma) \in B(\sigma, i)\}$
16 **if** $B'_j \subset B_j$ **then**
17 $B_{count} = B_j - B'_j$; $B_j = B'_j$
18 **foreach** $\sigma \in \Sigma$ **do**
19 $B(\sigma, count) = \{q \in B(\sigma, j) \mid q \in B_{count}\}$;
20 $B(\sigma, j) = \{q \in B(\sigma, j) \mid q \in B_j\}$
21 **if** $j \notin W(\sigma)$ *and* $0 < |B(\sigma, j)| \leq |B(\sigma, count)|$ **then**
22 $W(\sigma) = W(\sigma) \cup \{j\}$
23 **else**
24 $W(\sigma) = W(\sigma) \cup \{count\}$
25 $count = count + 1$;
26 splittable = false;
27 **foreach** $\sigma \in \Sigma$ **do**
28 **if** $W(\sigma) \neq \emptyset$ **then**
29 splittable=true;
30 Return partition blocks $B_1, ..., B_{count}$.

Algorithm 1: Kripke Structure Minimization

4. Correctness of Kripke Structure Minimization

In this section we give a rigorous but simple proof of the correctness of Algorithm 1. By means of a new induction argument, we have simplified the correctness argument compared with [1] and [3]. First let us establish termination of the algorithm by using an appropriate well-founded ordering for the main loop variant.

Definition 1. *Consider any pair of finite sets of finite sets* $A = \{A_1, ..., A_m\}$ *and* $B = \{B_1, ..., B_n\}$. *We define an ordering relation* $\leq$ *on* A *and* B *by* $A \leq B$ *iff* $\forall 1 \leq i \leq m$, $\exists 1 \leq j \leq n$ *such that* $A_i \subseteq B_j$. *Define* $A < B \iff A \leq B \ \& \ A \neq B$. *Clearly* $\leq$ *is a reflexive, transitive relation. Furthermore* $\leq$ *is well-founded, i.e. there are no infinite descending chains* $A_1 > A_2 > A_3 ...$, *since* $\emptyset$ *is the smallest element under* $\leq$.

Proposition 2. *Algorithm 1 always terminates.*

Proof. We have two cases for the termination of the algorithm as a result of the partition formed on line 1 of the algorithm: (1) when $n = |Q|$, and (2) when $n < |Q|$.

Consider the case when $n = |Q|$ then each block in the partition corresponds to a state of the given Kripke structure with a unique bit-label and hence in this case the algorithm will terminate on line 30 by providing the description of these blocks.

Now consider the case when $n < |Q|$. Then the waiting sets $W(\sigma)$ for all $\sigma \in \Sigma$ will be initialized on lines 7, 8 and the termination of the algorithm depends on proving the termination of the loop on line 10. Now $W(\sigma)$ is intialized by loading the block numbers of the split sets on line 8. There are only two possiblities after any execution of the loop. Let $W_m(\sigma)$ and $W_{m+1}(\sigma)$ represent the state of the variable $W(\sigma)$ before and after one execution of the loop respectively at any given time. Then either $W_m(\sigma) = W_{m+1}(\sigma) \cup \{i\}$ and no splitting has taken place and i is the deleted block number, or $W_m(\sigma) \cup \{j\} = W_{m+1}(\sigma) \cup \{i\}$ or $W_m(\sigma) \cup \{k\} = W_{m+1}(\sigma) \cup \{i\}$ where j and k represent the split blocks and one of them goes into $W_m(\sigma)$ if it has fewer incoming transitions. In either case $W_m(\sigma) > W_{m+1}(\sigma)$ by Definition 1. Therefore $W(\sigma)$ strictly decreases with each iteration of the loop on line 10. Since the ordering $\leq$ is well-founded, Algorithm 1 must terminate.

Now we only need to show that when Algorithm 1 has terminated, it returns the Nerode congruence $\equiv$ on states.

Proposition 3. *Let* P_i *be the partition (block set) on the ith iteration of Algorithm 1. For any blocks* $B_j, B_k \in P_i$ *and any states* $p \in B_j, q \in B_k$ *if* $j \neq k$ *then* $p \not\equiv q$.

Proof. By induction on the number i of times the loop on line 10 is executed.

Basis: Suppose $i = 0$ then clearly the result holds because each block created at line 1 is distinguishable by the empty string ϵ.

Induction Step: Suppose $i = m > 0$. Let us assume that the proposition holds after m executions of the loop.

Consider any $B_j, B_k \in P_m$. During the $m + 1$th execution of the loop on line 10 either block B_j is split into B'_j and B''_j or B_k is split into B'_k and B''_k but not both during one execution of the loop (due to line 17).

Consider the case when B_j is split then for any $p \in B_j$, either $p \in B'_j$ or $p \in B''_j$. But for any $p \in B_j$ and $q \in B_k$, $p \not\equiv q$ by the induction hypothesis. Therefore, for $p \in B'_j$ or $p \in B''_j$ $p \not\equiv q$. Hence the proposition is true for $m+1$th execution of the loop in this case.

By symmetry the same argument holds when B_k is split.

The following Lemma gives a simple, but very effective way to understand Algorithm 1. Note that this analysis is more like a temporal logic argument than a loop invariant approach. This approach reflects the non-determinism inherent in the algorithm.

Lemma 4. *For any states $p, q \in Q$, if $p \not\equiv q$ and initially p and q are in the same block $p, q \in B_{i_0}$ then eventually p and q are split into different blocks, $p \in B_j$ and $q \in B_k$ for $j \neq k$.*

Proof. Suppose that $p \not\equiv q$ and that initially $p, q \in B_{i_0}$ for some block B_{i_0}. Since $p \not\equiv q$ then for some $n \geq 0$, and $\sigma_1, \ldots, \sigma_n \in \Sigma$,

$$\lambda^*(p, \sigma_1, \ldots, \sigma_n) \neq \lambda^*(q, \sigma_1, \ldots, \sigma_n).$$

We prove the result by induction on n.

Basis Suppose $n = 0$, so that $\lambda(p) \neq \lambda(q)$. By line 1, $p \in B_p$ and $q \in B_q$ and $B_p \neq B_q$. So the implication holds vacuously.

Induction Step Suppose $n > 0$ and for some $\sigma_1, \ldots, \sigma_n \in \Sigma$,

$$\lambda^*(p, \sigma_1, \ldots, \sigma_n) \neq \lambda^*(q, \sigma_1, \ldots, \sigma_n).$$

(a) Suppose initially $\delta(p, \sigma_1) \in B(\sigma_1, \alpha)$ and $\delta(q, \sigma_1) \in B(\sigma_1, \beta)$ for $\alpha \neq \beta$.

Consider when $\sigma = \sigma_1$ on the first iteration of the loop on line 10. Clearly, $B(\sigma_1, \alpha), B(\sigma_1, \beta) \in W(\sigma)$ at this point. Choosing $i = \alpha$ and $j = i_0$ on this iteration then since $\delta(p, \sigma_1) \in B(\sigma_1, \alpha)$ we have

$$B'_{i_0} = \{t \in B_{i_0} \mid \delta(t, \sigma_1) \in B(\sigma_1, \alpha)\} \subset B_{i_0}$$

This holds because $q \in B_{i_0}$ but $\delta(q, \sigma_1) \in B(\sigma_1, \beta)$ and $B(\sigma_1, \alpha) \neq B(\sigma_1, \beta)$ so $B(\sigma_1, \alpha) \cap B(\sigma_1, \beta) = \emptyset$ and hence $q \notin B'_{i_0}$. Therefore p and q are split into different blocks on the first iteration so that $p \in B'_{i_0}$ and $q \in B_{i_0} - B'_{i_0}$.

By symmetry, choosing $i = \beta$ and $j = i_0$ then p and q are split on the first loop iteration with $q \in B'_{i_0}$ and $p \in B_{i_0} - B'_{i_0}$.
(b) Suppose initially $\delta(p, \sigma_1), \delta(q, \sigma_1) \in B(\sigma_1, \alpha)$ for some α. Now

$$\lambda^*(\, \delta(p, \sigma_1), \sigma_2, \ldots, \sigma_n \,) \neq \lambda^*(\, \delta(q, \sigma_1), \sigma_2, \ldots, \sigma_n \,).$$

So by the induction hypothesis, eventually $\delta(p, \sigma_1)$ and $\delta(q, \sigma_1)$ are split into different blocks, $\delta(p, \sigma_1) \in B_\alpha$ and $\delta(p, \sigma_1) \in B_\beta$. At that time one of B_α or B_β is placed in a waiting set $W(\sigma)$. Then either on the same iteration of the loop on line 10 or on the next iteration, we can apply the argument of part (a) again to show that p and q are split into different blocks.

Observe that only one split block is loaded into $W(\sigma)$ on lines 21-24. From the proof of Lemma 4 we can see that it does not matter logically which of these two blocks we insert into $W(\sigma)$. However, by choosing the subset with fewest incoming transitions we can obtain a worst case time complexity of order $O(kn\ log_2\ n)$, as we will show.

Corollary 5. *For any states $p, q \in Q$, if $p \not\equiv q$ then p and q are in different blocks when the algorithm terminates.*

Proof. Assume that $p \not\equiv q$.
(a) Suppose at line 3 that $n = |Q|$. Then initially, all blocks B_i are singleton sets and so trivially p and q are in different blocks when the algorithm terminates.
(b) Suppose at line 3 that $n < |Q|$.
(b.i) Suppose that p and q are in different blocks initially. Since blocks are never merged then the result holds.
(b.ii) Suppose that p and q are in the same block initially. Since $p \not\equiv q$ then the result follows by Lemma 4.

5. Complexity Analysis

Let us consider the worst-case time complexity of Algorithm 1.

Proposition 6. *If $\mathcal{K}$ has n states and Σ has k input symbols then Algorithm 1 has worst case time complexity $O(kn \log_2 n)$.*

Proof. Creating the initial block partition on line 1 requires at most $O(n)$ assignments. The block subpartitioning in the loop on line 3 requires at most $O(kn)$ moves of states. Also the the initialisation of the waiting lists $W(\sigma)$ in the loop on line 7 requires at most $O(kn)$ assignments.

Consider one execution of the body of the loop starting on line 10, i.e. lines 13 - 29. Consider any states $p, q \in Q$ and suppose that $\delta(p, \sigma) = q$ for some $\sigma \in \Sigma$. Then the state p can be: (i) moved into B'_j (line 15), (ii) removed from B_j (line 17), or (iii) moved into $B(\sigma, i)$ or $B(\sigma, count)$ (lines 19, 20) if, and only if, a block i is being removed from $W(\sigma)$ such that $q \in B(\sigma, i)$ at that time. (Such a block sub-partition $B(\sigma, i)$ can be termed a *splitter* of q.)

Now each time a block i containing q is removed from $W(\sigma)$ its size is less than half of the size when it was originally entered into $W(\sigma)$, by lines 21-24. So i can be removed from $W(\sigma)$ at most $O(log_2\ n)$ times. Since there are at most k values of σ and n values of p, then the total number of state moves between blocks and block sub-partitions is at most $O(kn\ log_2\ n)$.

6. Conclusions

We have given an algorithm for the minimization of deterministic Kripke structures with worst case time complexity $O(kn \log_2 n)$. We have analysed the correctness and performance of this algorithm. An efficient implementation of this algorithm has been developed which confirms the run-time performance theoretically predicted in Section 5. This research has been supported by the Swedish Research Council (VR), the Higher Education Commission of Pakistan (HEC), as well as EU projects HATS FP7-231620, and MBAT ARTEMIS JU-269335.

References

[1] Berstel, J., Boasson, L., Carton, O., Fagnot, I., Oct. 2010. Minimization of Automata. ArXiv e-prints.

[2] Bustan, Grumberg, 2003. Simulation-based minimization. ACMTCL: ACM Transactions on Computational Logic 4.

[3] Hopcroft, J. E., 1971. An n log n algorithm for minimizing states in a finite automaton. In: Kohavi, Z., Paz, A. (Eds.), Theory of Machines and Computations. Academic Press, pp. 189–196.

[4] Meinke, K., Sindhu, M. A., 2011. Incremental learning-based testing for reactive systems. In: Gogolla, M., Wolff, B. (Eds.), Tests and Proofs. Vol. 6706 of Lecture Notes in Computer Science. Springer, pp. 134–151.